PROGRAMMING

TUTORIAL / PRACTIAL 01

SESSION 01

Part 1

– Review our approach to try and make things easier
– Introduce todays problem
– Discuss JavaScript basics
– Talk about HTML5 and how to implement and run JavaScript
– Enable/viewing debug information in a webpage
– Review flowcharts
– Review Pseudocode

Part 2

– JavaScript Activity

MAKE THINGS EASIER FOR YOU

To try to help make things easier while you're getting used to programming – we will try to avoid (in the first instance) going into too much detail about the various aspects being discussed or explored.

However, as things start to become more familiar you'll find that we will start to revisit several previous points in order to better expand your understanding and knowledge. These deeper dives will help you delve into the more complex underpinning reasons and may require you to adapt or redefine earlier definitions.

It's suggested that you try to build and refine your own internal definitions and knowledge of the various aspects being discussed as we explore each so that you can more easily use, recall and explain the purpose, benefit or in some situations the possible constraints/ or disadvantage of using certain methods, instructions or solutions.

TODAY'S PROBLEM STATEMENT

Design, test, code and debug a JavaScript program that presents (displays) a random number between 0 and 100 (inclusive) and asks the user whether the next number will be greater or less than the number already shown.

JAVASCRIPT INTRODUCTION

In general, JavaScript (JS) is a scripting languages that's principally used to enhance HTML web pages. JavaScript is an interpreted language meaning that it doesn't need to be compiled. Because it is typically not "platform" specific the same JavaScript code can be executed or used on many different platforms (e.g. Mac, PC, Linux, Unix, Android, iOS, etc).

The browser (e.g. Chrome, Firefox, Safari, Edge, Explorer, etc.) is in charge of rendering the web page and executing any JavaScript code. Programmers can use JavaScript to create fully interactive user experiences (including music, video and even 3d games).

JavaScript can (but is not limited to) be programmed to:

- React to Events
- Display Special FXs
- Accept and Validate Input
- Detect and Respond to User and Hardware Settings
- Output Results
- Manipulate HTML objects

RUNNING JAVASCRIPT

JavaScript does not need to be compiled, you only need put it between <script> and </script> tags in a standard HTML document, save then open it in a browser.

Suggested Browser: Chrome

- Free
- Runs on UH machines and supports all the common operating system platforms
- Provides excellent support (will be explored in more detail later)

Suggested Editor: Notepad ++

- Free
- Installed on UH machines
- Supports syntax highlighting, folding, search & replace, multiple tabs for multiple documents)

UNDERSTANDING DEPENDENCY

What is dependency in programming?

How does the computer recognise that dependency?

How does a programmer recognise that dependency?

HTML5 BASIC EXAMPLE 1/4

1. **<!doctype html>** *<!-- defines the type of document being read by the browser -->*
2. **<html lang="en">** *<!-- defines the start of html and the language being used -->*

3. **<head>** *<!-- defines the start of the code block that supports the use and function of the page -->*
   <title>4COM1037</title> *<!-- used to declare the name of the page -->*
4. **<script>** *<!– tells the browser that this is a JavaScript code block -->*
   </script> *<!– tell the browser that this is the end of the JavaScript code block -->*
   </head> *<!– tells the browser that this is the end of the head block -->*

5. **<body>** *<!– tells the browser that this is the start of the html body; all the bits to be rendered -->*
   </body> *<!– tells the browser that this is the end of the html body; nothing else to render -->*

</html> *<!– tells the browser that this is the end of the entire html page -->*

HTML5 BASIC EXAMPLE 2/4 (CODE BLOCKS)

```
1. <!doctype html> <!-- defines the type of document being read by the browser -->
2. <html lang="en"> <!-- defines the start of html and the language being used -->

3.     <head> <!-- defines the start of the code block that supports the use and function of the page -->
           <title>4COM1037</title> <!-- used to declare the name of the page -->
4.         <script> <!-- tells the browser that this is a JavaScript code block -->
           </script> <!-- tell the browser that this is the end of the JavaScript code block -->
       </head> <!-- tells the browser that this is the end of the head block -->

5.     <body> <!-- tells the browser that this is the start of the html body; all the bits to be rendered -->
       </body> <!-- tells the browser that this is the end of the html body; nothing else to render -->

</html> <!-- tells the browser that this is the end of the entire html page -->
```

HTML5 BASIC EXAMPLE 3/4 (JAVASCRIPT)

```
<!doctype html> <!-- defines the type of document being read by the browser -->
<html lang="en"> <!-- defines the start of html and the language being used -->

    <head> <!-- defines the start of the code block that supports the use and function of the page -->
        <title>4COM1037</title> <!-- used to declare the name of the page -->
        <script> <!-- tells the browser that this is a JavaScript code block -->
            function fnHello() {
                alert("Hello World");
            }
        </script> <!-- tell the browser that this is the end of the JavaScript code block -->
    </head> <!-- tells the browser that this is the end of the head block -->

    <body> <!-- tells the browser that this is the start of the html body; all the bits to be rendered -->
    </body> <!-- tells the browser that this is the end of the html body; nothing else to render -->

</html> <!-- tells the browser that this is the end of the entire html page -->
```

HTML5 BASIC EXAMPLE 4/4 (INTERACTION)

```
<!doctype html> <!– defines the type of document being read by the browser –>
<html lang=“en”> <!– defines the start of html and the language being used –>

    <head> <!– defines the start of the code block that supports the use and function of the page –>
        <title>4COM1037</title> <!– used to declare the name of the page –>
        <script> <!– tells the browser that this is a JavaScript code block –>
            function fnHello() {
                alert(“Hello World”);
            }
        </script> <!– tell the browser that this is the end of the JavaScript code block –>
    </head> <!– tells the browser that this is the end of the head block –>

    <body> <!– tells the browser that this is the start of the html body; all the bits to be rendered –>
        <button id="button1" onclick="fnHello();">Hello</button>
    </body> <!– tells the browser that this is the end of the html body; nothing else to render –>

</html> <!– tells the browser that this is the end of the entire html page –>
```

ERROR RAGE (A BIT LIKE ROAD RAGE)

Programming errors can be/are:

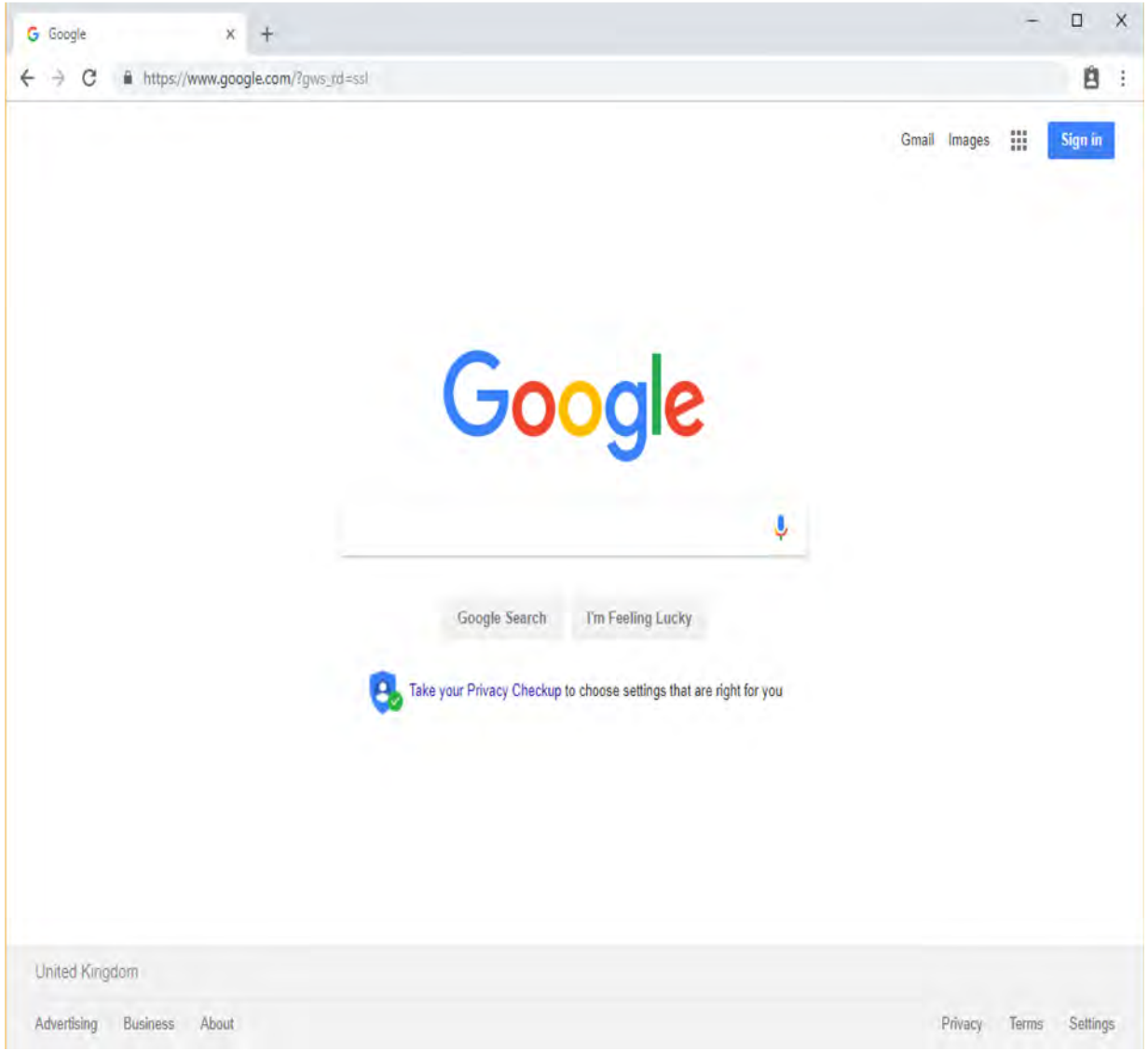
- Frustrating
- Annoying
- Time Consuming

Consider the following two questions?

1. What other negative things are they?
2. Obviously they don't have any positive value do they?

DIAGNOSING JAVASCRIPT ERRORS

If you're experiencing unexpected behaviour with your program's functionality (e.g. your alert box does not appear) you may have some kind of errors. In order to better debug you code it will likely help if you can see the error details in your browser.

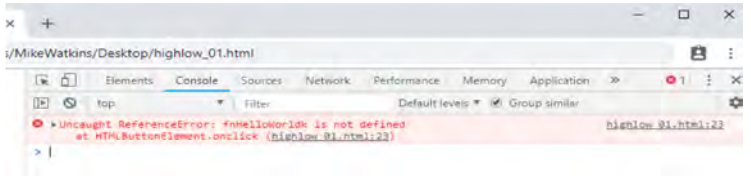


1. Open the Console

Open the HTML page you are experiencing a problem with and in Chrome click the Control icon (3 Vertical Dots, Top Right) and select: More Tools > Developer Tools

The Console (as well as a number of other tools) normally launches on the right of the browser.

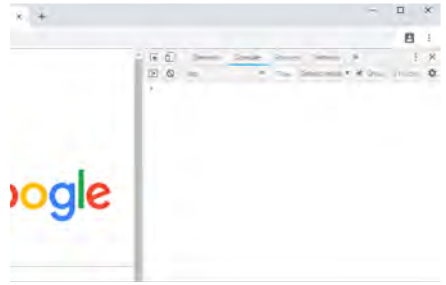
DIAGNOSING JAVASCRIPT ERRORS



1. Identify the Error

2. No errors in the console? Try reloading the page; errors may be generated when the page loads or the action causing the issue is detected.

3. The console will provide you with the error type, the location of the error and the line number.

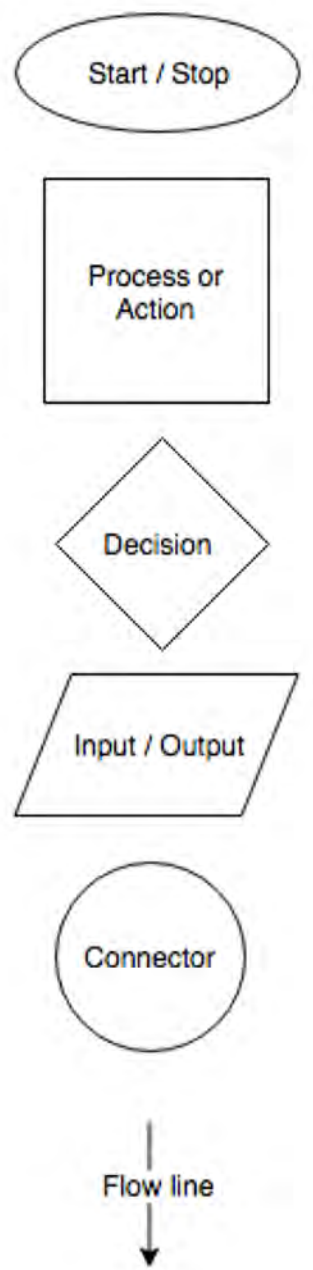


FLOWCHART

(HTTPS://WWW.DRAW.IO/ A FREE ONLINE TOOL TO CREATE FLOWCHARTS)

Flowcharts are an excellent way to represent an algorithm (a.k.a. a solution to a problem). In programming terms, Flowcharts are a type of diagram that can be used to review, design, test, document or analyse the overall logic of a solution.

For the purpose of this module we will utilise the 6 different shapes (start/stop, process or action, decision, input/output, connector and flowline).



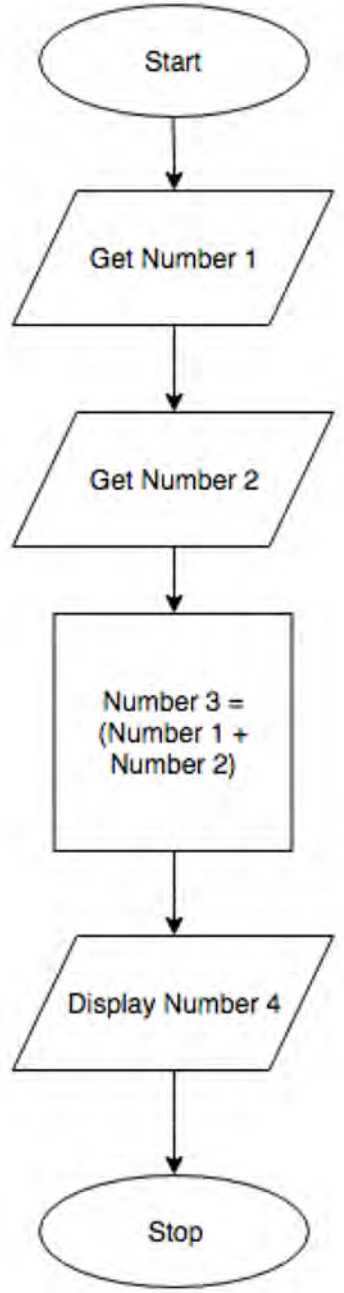
Setting some standards:

- All programs must have only 1 Start.
- Individual Process/Action boxes must have only 1 incoming and 1 outgoing flow line.
- Individual Decision boxes must have only 1 incoming and 2 outgoing flow lines (Yes and No).
- Individual Input/Output boxes must have only 1 incoming and 1 outgoing flow line.
- Individual Connectors can have multiple incoming but only 1 outgoing flow line and must be numbered (connectors are used to link flow lines and help minimise the possible complexity of a design).
- Flow lines can connect to other flow lines (this is like merging traffic into a lane – they will now all travel in the same direction)
- Flow lines should be orientated along a horizontal and/or vertical axis only (try to avoid diagonals)

ADDING TWO NUMBERS

Review the flowchart:

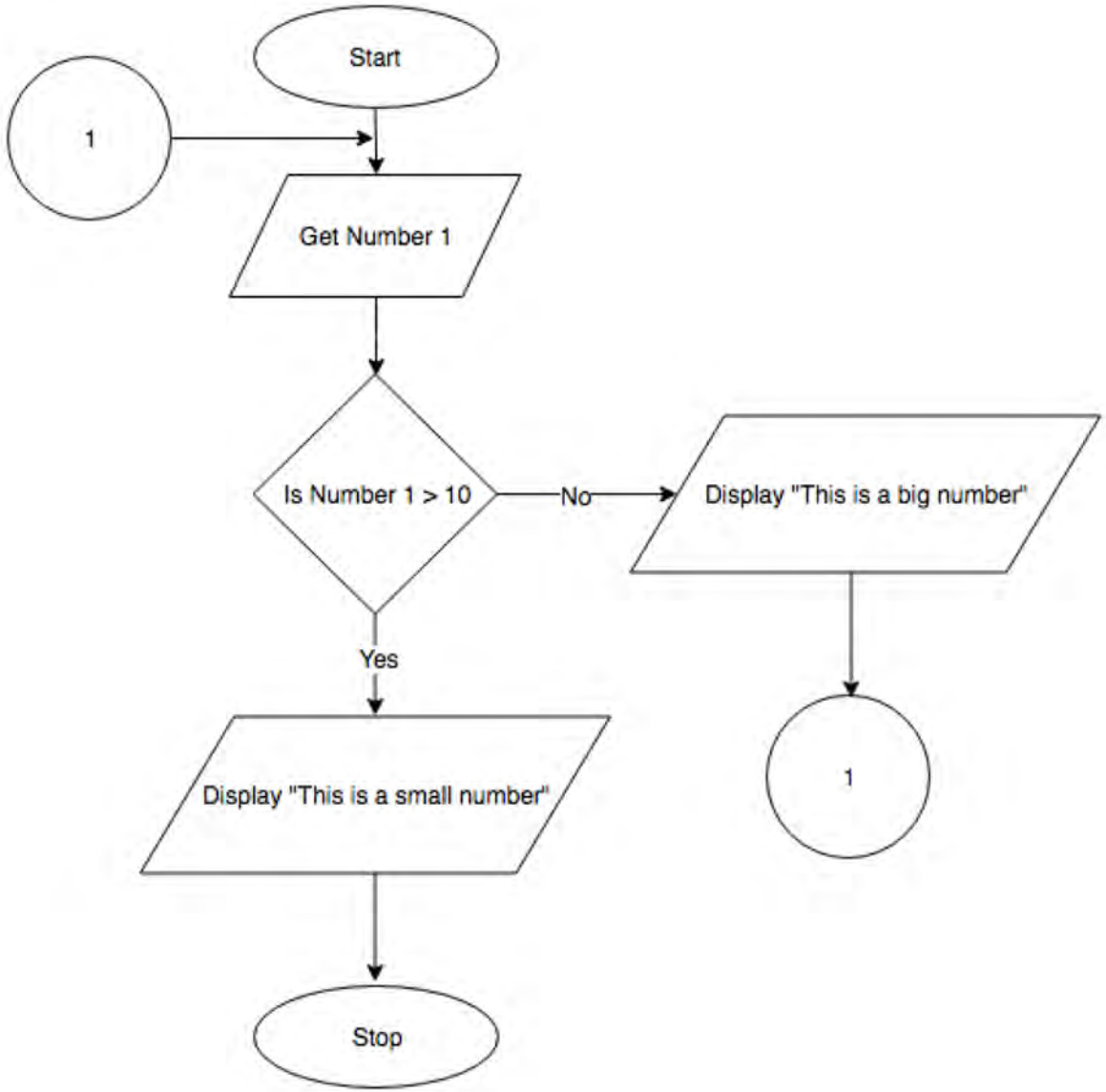
- What does it do?
- What information does it need?
- Where can it get that information?
- What does it do with the information?
- Can it be improved?



WHAT DOES THIS ALGORITHM DO?

Review the flowchart:

- What does it do?
- What information does it need?
- Where can it get that information?
- What does it do with the information?
- Can it be improved?



```
START
    SET total = 0
    SET grade = 1
    WHILE grades <= 10
        INPUT nextgrade
        SET total = total + net grade
    END
    SET classaverage = total / 10
    DISPLAY classaverage
STOP
```

PSEUDOCODE

Pseudocode is a simple, informal text-based language with a very limited vocabulary that programmers can use to help develop algorithms. Pseudocode is best suited to help design functions or actions.

Pseudocode rules are generally straightforward. All statements showing a "dependency" are indented. These include: while, do, for, if, start, etc.

1. What does this algorithm do?
2. Can it be improved?

Bracket	Purpose
( ) round or parentheses	Used to define what information is being passed or accepted by a function. Used to help define the order in which a calculation is processed.
{ } curly or braces	Used to indicate the start and end of a dependent code block
[ ] square	Used to indicate an index number; index numbers are basically a pointer to a position in a list
< > angle	Used in HTML to differentiate a TAG, e.g. <html> </html>

Operation	Description
=	A = B, set the variable A to the value of B
==	A == B, compare the value A to the value B
<=	A <= B, compare whether A is less than or equal to B
>=	A >= B, compare whether A is greater than or equal to B
<	A < B, compare whether A is less than B
>	A > B, compare whether A is greater than B

PROGRAMMING YOUR ALGORITHM

A working algorithm generally means you have got an overall resolution and understand the order of steps needed to solve the problem,

Converting an algorithm to a program takes practice, patience and clear thinking (oh, and lots of testing your code)

PROGRAMMING YOUR ALGORITHM

- Don't try to build the entire solution in one instance – break it up in to bits first!!
- Focus on the easy to finish bits, research and create small prototypes to test ideas and improve understanding.
- Build and test a bit until you are happy it is DONE and has been tested and works.
- Move onto the next bit and repeat.
- Only add (or change a small bit) at a time.
- Save and test your code each time you make a change!!
- Keep older copies of your code by renaming the source file with a version number (e.g. highlow_01.html).
- Ensure you comment all your lines, if you don't know what it does and why you will have a problem.
- Use old copies as a source of reference (build your own library of simple, small working examples).

WARNING

WHAT WILL HAPPEN IF YOU DON'T TEST EACH BIT YOU CHANGE AS YOU GO?

SOLVE STUFF

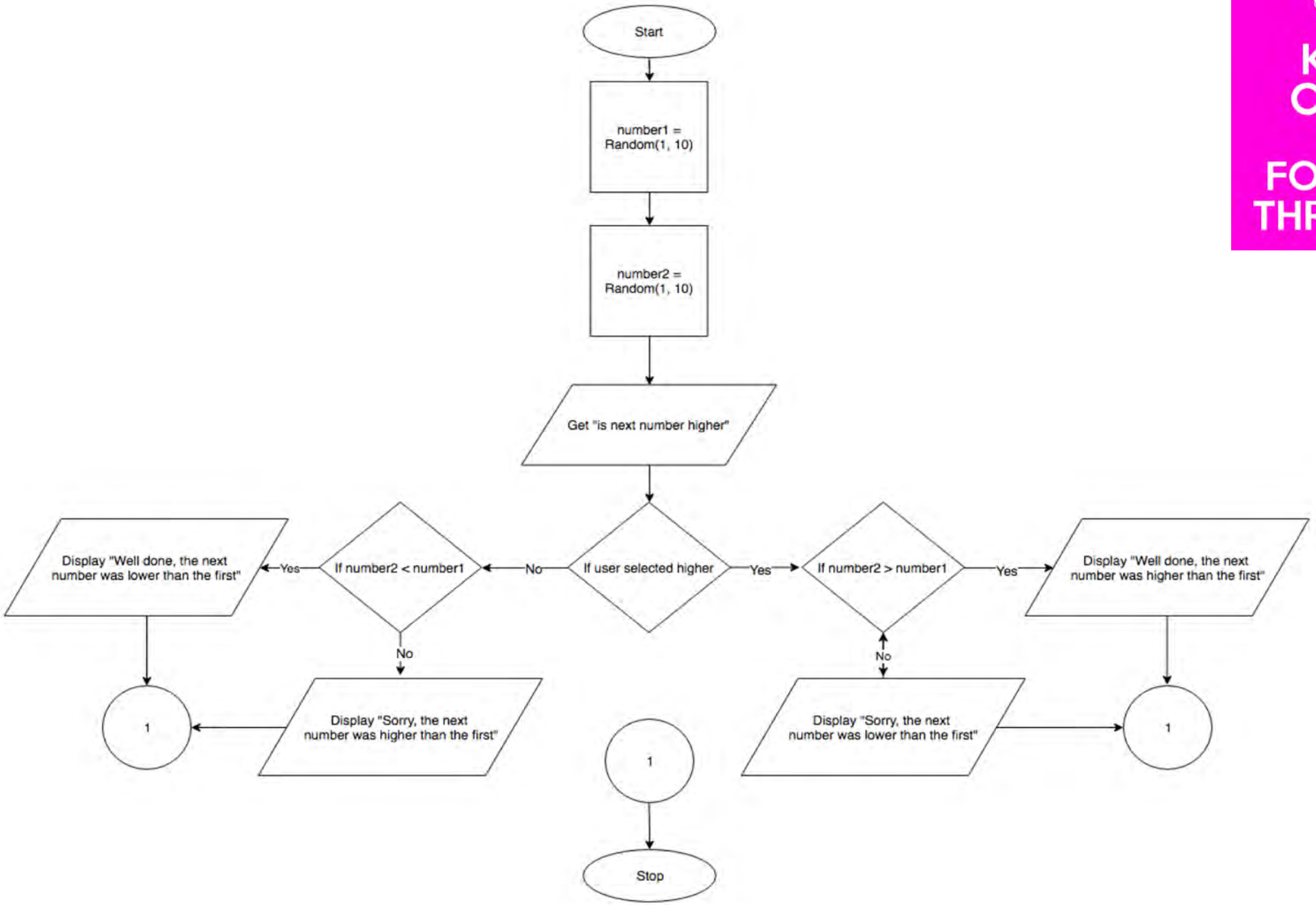
JAVASCRIPT PROBLEMS

HIGH/LOW PROBLEM TASK 1 (10/15MINS)

Design, test, code and debug a JavaScript program that presents (displays) a random number between 0 and 100 (inclusive) and asks the user whether the next number will be greater or less than the number already shown.

1. **As either an individual or in groups of 2 discuss the problem, consider the following:**
   - *What information does the program need?*
   - *What information can the program provide?*
   - *When and where will that information be provided or calculated?*
     The purpose of this stage is to help you start to build a logical approach to problem solving. Start with simple questions like, "what has to happen first", "okay, now what?", "then what?", etc – repeat this process until you feel you have broken down all the steps needed to solve this problem.

2. **Got a possible idea for your algorithm on the steps and the order needed?**
   - *Based on your notes (or from memory) try to construct a flowchart (either on paper or via the DrawIO site)*
   - *Test your flowchart by carefully walking through each step, use this time to try to improve or refine your solution so it works and is as simple as possible*



PRACTICAL WORK

01

Download and try to complete the following tutorials in order:

- highlow_tutorial_01.html
- highlow_tutorial_02.html
- highlow_tutorial_03.html
- highlow_tutorial_04.html
- highlow_tutorial_05.html

02

Open in your preferred editor then read and review the code. Each tutorial contains a number of tasks that are specifically designed to get progressively more challenging.

03

This weeks challenge is to complete each of the tutorials by the start of your next session; we will be building on that experience so try not to miss these opportunities